Refine Search

Search Results -

Term	Documents
((3.AB.) AND (24.AB.)).USPT.	27
(L3.AB. AND L24.AB.).USPT.	27

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L39

Refine Search

Recall Text | Clear | Interrupt

Search History

DATE: Sunday, March 21, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L39	l3.ab. and l24.ab.	27	L39
L38	l1 and l3 and L37	46	L38
L37	l2 near4 l24	714	L37
L36	L35 and l3.ab.	18	L36
L35	l1 and l2 and l3 and l4 and l13 and l24	217	L35
L34	l1.ab. and L33	3	L34
L33	l4 and L32	37	L33
L32	l13 and l24 and L29	46	L32
L31	l13 and l24 and L27	3	L31
L30	l7 and l24 and L29	0	L30
L29	l3.ab.	1286	L29
L28	l7 and l24 and L27	0	L28
L27	schedul$.ab. and l3.ab.	32	L27

h e b b cg b e e ch

L26	l1.ab. and l3 and l13 and L25	6	L26
L25	l4 same L24	14074	L25
L24	priorit$	141960	L24
L23	l21 and L15	0	L23
L22	l15 and L21	0	L22
L21	workload	5397	L21
L20	l15 and L19	0	L20
L19	I/O	75594	L19
L18	l15 and L17	0	L18
L17	I/O	75594	L17
L16	l14 and L15	0	L16
L15	dilay$	33	L15
L14	l6 and L13	122	L14
L13	dispat$	16559	L13
L12	l6 and l7	0	L12
L11	l6 and l7 and L8	0	L11
DB=TDBD; PLUR=YES; OP=ADJ			
L10	l6 and l7	0	L10
L9	l6 and l7 and L8	0	L9
L8	delay$	6044	L8
L7	dispat$	550	L7
DB=USPT; PLUR=YES; OP=ADJ			
L6	l1 and l2 and L5	317	L6
L5	L4 same l3	1839	L5
L4	check$ or compar$	1662921	L4
L3	client$1 near5 request$	9332	L3
L2	transaction$1	88023	L2
L1	schedul$	72669	L1

END OF SEARCH HISTORY

h e b b cg b e e ch

L23: Entry 6 of 19 File: USPT Aug 20, 2002

DOCUMENT-IDENTIFIER: US 6438629 B1
TITLE: Storage device buffer access control in accordance with a monitored latency parameter

Brief Summary Text (31):
The latency monitor may comprise a setting register, a counter, and a high latency signaller. The setting register holds a value representing a threshold delay in granting a given client access to the buffer. The counter counts an amount of time elapsing from a reset time. The reset time may be the last time the given client was granted access, or the time at which the given client requests access to the buffer. The high latency signaller signals to the buffer access controller when the threshold delay has been reached by the counter. To render the latency monitor easily configurable, the setting register may be software programmable.

Brief Summary Text (32):
The buffer channel access controller may comprise a priority-based arbitrator which assigns to an access request from the given client a default priority when the threshold delay has not been reached by the counter, and a higher priority when the threshold delay has been reached by the counter. The priority-based arbitrator may comprise a fixed-priority encoder.

Brief Summary Text (33):
In accordance with another aspect of the present invention, a hard disk buffer arbitration subsystem may be provided which responds to client requests to provide, to a microprocessor or microcontroller, disk input/output processes, and hard disk control processes, access to the buffer. A latency monitor is provided for monitoring a latency parameter indicative of the buffer access latency for a given client. A buffer access controller controls when the given client is given access to the buffer in accordance with the latency parameter. The buffer may comprise a random access memory.

Brief Summary Text (34):
The latency monitor may comprise a setting register, a counter, and a high latency signaller. The setting register holds a value representing a threshold delay in granting a given client access to the buffer. The counter counts an amount of time elapsing from a reset time--e.g., a time at which a buffer access request was last granted to the given client. The high latency signaller signals to the buffer access controller when the threshold delay has been reached by the counter. The setting register may be software programmable. The buffer channel access controller may comprise a priority-based arbitrator which assigns to an access request from the given client a default priority when the threshold delay has not been reached by the counter and a higher priority when the threshold delay has been reached by the counter. The priority-based arbitrator may comprise a fixed-priority encoder.

Detailed Description Text (8):
As shown in FIG. 2, hard disk central controller 26 comprises a microprocessor 34, a host I/O interface 36, a disk controller/servo 38, a cache manager 40, a disk I/O interface 42, and a memory arbitrator/controller 44. A parallel bus 33 is provided to which each of the aforementioned units is connected. Each of the units, forming a part of hard disk central controller 26, may communicate with another unit via

parallel bus 33, to carry out typical data management operations associated with a hard disk device. In the illustrated embodiment, parallel bus 33 is of a dimension sufficient (e.g., 16 bits, 32 bits, or more) to avoid unwanted delays such as in accessing memory buffer 32.

Detailed Description Text (9):
As shown in FIG. 2, each of microprocessor 34, host I/O interface 36, disk controller/servo 38, cache manager 40, and disk I/O interface 42 comprises a respective one of several single bit direct connection lines 46a-46e connected directly to memory arbitrator/controller 44. Each of these lines 46a-46e is an arbitration request line. Each of the illustrated controller units sends an arbitration request via its respective arbitration request line to memory arbitrator/controller 44 in order to obtain access to memory buffer 32. Memory arbitrator/controller 44 serves as a buffer access determining mechanism, and determines when a given client of memory buffer 32 is to be given access to buffer channel 31. Memory buffer 32 has a limited storage capacity and comprises a buffer channel 31 having a limited data transfer rate, thus defining a maximum access bandwidth to memory buffer 32.

Detailed Description Text (10):
The various controller units provided as part of hard disk central controller 26 manage the exchange of data between host computer 12 and the auxiliary hard disk storage device 10 and further manage the exchange of data within hard disk storage device 10 which serves as a disk storage memory subsystem. Accordingly, the management process will translate memory access requests it receives from application 22 or operating system 24 to memory access requests corresponding to actual addresses located within memory buffer 32. If disk data is requested not yet having an actual address located within the memory buffer 32, or the destination is not within memory buffer 32, the process will trigger the operation of disk I/O interface 42 to load the requested data in buffer 32 from the disk media.

Detailed Description Text (12):
Disk controller/servo further controls the read/write head servo mechanism, and, for this purpose, stores and retrieves servo data in memory buffer 32-via memory arbitrator/controller 44. Cache manager 40 sets up instructions within memory buffer 32 to be later accessed and utilized by disk controller/servo 38 and by disk I/O interface 42, and accordingly provides direction to each of disk I/O interface 42 and disk controller/servo 38 to control the exchange of data between memory buffer 32 and the disk media.

Detailed Description Text (13):
Disk I/O interface 42 handles reads from and writes to the disk media. Disk I/O interface 42 stores disk data, read from the disk media, in memory buffer 32.

Detailed Description Text (14):
Host I/O interface 36 hands off data to and from host computer 12, and stores host data within memory buffer 32 via memory arbitrator/controller 44.

Detailed Description Text (16):
The illustrated controller units which require access to memory buffer 32 include microprocessor 34, disk controller/servo 38, cache manager 40, disk I/O interface 42, and host I/O interface 36. Other controller units (not shown) could be provided as well, thus increasing the demand for access to memory buffer 32 and the accompanying need to manage such access with the use of an arbitration scheme. For example, one or more additional units may be provided which perform operations on data within the memory buffer, e.g., for facilitating RAID operations, search routines, and so on. Accordingly, in the illustrated embodiment, a memory arbitrator/controller 44 is provided to manage the conflicting access requests that are encountered in such a hard disk controller.

Detailed Description Text (18):
Memory controller 52, comprises, among other elements (not specifically shown), a memory refresh timer 54 and a grant indicator 56. Memory controller 52 performs such functions as intercepting client requests for access to memory buffer 32, and notifying clients when access has been granted to memory buffer 32 by use of grant indicator 56. Memory controller 52 controls the storing of data to memory buffer 32 via buffer channel 50.

Detailed Description Text (26):
Requestor signal E corresponds to host data to be stored in memory buffer 32 per the request of host I/O interface 36 via arbitration request line 46e.

Detailed Description Text (27):
Requestor signal F pertains to disk data to be read from the disk media and stored in memory buffer 32, and corresponds to an access request made by disk I/O interface 42 via arbitration request line 46d.

Detailed Description Text (33):
As noted above, the requester signal F corresponds to disk data either to be written to or read from the disk media per the access request of disk I/O interface 42. Memory controller 52 may be programmed to change the values in one or more of the setting registers dynamically to accommodate different operation modalities of memory arbitrator/controller 44. For example, to give the application 22 the appearance of quicker disk data access when a disk read operation is performed, a relatively small threshold value may be set in disk data setting register 66a. On the other hand, when disk data is to be written to the disk media, from memory buffer 32, a higher threshold may be appropriate. Accordingly, when a disk write is Being performed as opposed to a disk read, memory controller 52 may be programmed to change the value in disk data setting register 66a to a higher value.

CLAIMS:

9. The hard disk buffer access priority subsystem according to claim 7, wherein said buffer channel access controller comprises a priority-based arbitrator which assigns to an access request from said given client a default priority when said threshold delay has not been reached by said counter and a higher priority when said threshold delay has been reached by said counter.

First Hit Fwd Refs

Generate Collection Print

L26: Entry 2 of 6 File: USPT Jan 7, 2003

DOCUMENT-IDENTIFIER: US 6505250 B2
TITLE: Apparatus and method for scheduling and dispatching queued client requests within a server in a client/server computer system

Abstract Text (1):
An apparatus for scheduling and dispatching client requests for execution by a server object in a heterogeneous object-oriented client/server computing environment, the apparatus comprising: a request-holding buffer having an input connected to a communications channel which channels the client requests to the apparatus, and an output; a plurality of parallel execution threads connected to the output of the buffer; and a scheduling means for distributing client requests stored in the buffer to the plurality of execution threads, characterized in that: the scheduling means places client requests held in the buffer in priority order based on a priority determining rule which takes into account the state of the plurality of execution threads and the nature of each of the held requests.

Brief Summary Text (2):
The invention relates to the field of client/server (also known as "distributed") computing, where one computing device ("the client") requests another computing device ("the server") to perform part of the client's work.

Brief Summary Text (5):
The benefits of client/server computing have been even further enhanced by the use of a well-known computer programming technology called object-oriented programming (OOP), which allows the client and server to be located on different (heterogeneous) "platforms". A platform is a combination of the specific hardware/software/operating system/communication protocol which a machine uses to do its work. OOP allows the client application program and server application program to operate on their own platforms without worrying how the client application's work requests will be communicated and accepted by the server application. Likewise, the server application does not have to worry about how the OOP system will receive, translate and send the server application's processing results back to the requesting client application.

Brief Summary Text (8):
When the client computer 10 wishes to make a request for the server computer 20's services, the first application program 40 informs the first logic means 50 of the service required. It may for example do this by sending the first logic means the name of a remote procedure along with a list of input and output parameters. The first logic means 50 then handles the task of establishing the necessary communications with the second computer 20 with reference to definitions of the available communications services stored in the storage device 60. All the possible services are defined as a cohesive framework of object classes 70, these classes being derived from a single object class. Defining the services in this way gives rise to a great number of advantages in terms of performance and reusability.

Brief Summary Text (13):
FIG. 2 shows a conventional architecture for such a system. Once client requests find their way through the ORB 21 and into the server, the ORB finds a particular server object capable of executing the request and sends the request to that server

object's object adapter 22 (also defined by OMG standard) where it is stored in the object adapter's buffer to await processing by the server object. The buffer is a First-In-First-Out queue, meaning that the first request received in the buffer at one end thereof is the first to leave out the other end. The server object has a plurality of parallel execution threads (23a, 23b, 23c) upon any of which it can run an instance of itself. In this way, the server object is able to process plural requests at the same time. The object adapter 22 looks to see which of the parallel execution threads is ready to process another request and assigns the request located at the end of the buffer to the next available execution thread. This is explained in the above-mentioned U.S. Patent as a "dispatching" mechanism whereby the server dispatches queued requests to execution threads.

Brief Summary Text (14):
One major problem with this prior architecture is that it is not possible to obtain a predictable response time for the execution of a client request. That is, a particular client request could be sitting in a server object's object adapter queue 22 behind a large number of other requests, or, at another time, the particular client request could be the only request in the queue. The client that is waiting for an answer cannot predict when a response will be received from the server object. Another problem is that a very important client request may have to wait behind many not so important requests in the object adapter queue.

Brief Summary Text (17):
According to one aspect, the present invention provides an apparatus for scheduling and dispatching client requests for execution by a server object in a heterogeneous object-oriented client/server computing environment, the apparatus comprising: a request-holding buffer having an input connected to a communications channel which channels the client requests to the apparatus, and an output; a plurality of parallel execution threads connected to the output of the buffer; and a scheduling means for distributing client requests stored in the buffer to the plurality of execution threads, characterized in that: the scheduling means places client requests held in the buffer in priority order based on a priority determining rule which takes into account the state of the plurality of execution threads and the nature of each of the held requests.

Brief Summary Text (20):
According to a second aspect, the present invention provides a method of scheduling and dispatching client requests for execution by a server object in a heterogeneous object-oriented client/server computing environment, comprising the steps of: determining information about each of a plurality of queued incoming client requests; determining information about each of a plurality of parallel execution threads of the server object; applying a priority determining rule to the information obtained in said determining steps; and scheduling the order of dispatch from the queue of the plurality of queued requests based on the results of said applying step.

Brief Summary Text (22):
Thus, with the present invention, queued client requests can be processed in a much more efficient and controllable manner, greatly enhancing the predictability of processing result which is returned to the client. High priority client requests can be processed before lower priority requests and workload management amongst the execution threads can be effected, to provide highly efficient and predictable processing of the queued requests.

Detailed Description Text (2):
In the preferred embodiment of FIG. 3, requests received at the server process from client processes are first received by the server's ORB 31. ORB 31 then passes on requests destined to a particular server object to that server object's object adapter 32. This server object has a number of parallel execution threads 33a, 33b and 33c where different instances of the server object can be running in parallel,

in order to execute a large number of client requests. This is all analogous to the prior art of FIG. 2 that was described above.

Detailed Description Text (4):
In the example that will be described hereinbelow to illustrate the operation of this preferred embodiment, the server object will represent a bank account. Thus, the various requests that are queued in object adapter 32 are requests to access a particular bank account. One queued request is from a client ATM (automated teller machine) to withdraw funds from this account. This request is from the person owning the account who wishes to withdraw some funds. A second queued request is from a direct deposit salary payer client. This request is from the account owner's employer and the employer is adding the employer's monthly salary into the account owner's bank account. A third queued request is from another client ATM to check the balance of the account. This request is from the account owner's wife, who is on the other side of town from the owner at another client ATM machine. A fourth queued request is a direct debit request from the electricity company that supplies electricity to the account owner's household. The request is a debit of the account for the amount of the monthly electricity bill.

Detailed Description Text (5):
The priority determining unit 34 operates according to a programmed rule in order to determine the priority of the queued requests in object adapter 32 that are awaiting execution by the server object. For example, one part of the rule is that requests to check the balance of the account should be given a low priority value, since the reply to this request will be more informative to the client if other pending requests are executed first. That is, if a large amount of money is going to be debited from the account by a direct debit, it is better that the person requesting the balance of the account be given the balance after the direct debit rather than before the direct debit. This gives a more current version of the balance to the person requesting the balance.

Detailed Description Text (6):
Another part of the rule is that if threads 33a, 33b and 33c are heavily loaded (are performing a high level of processing as compared to normal) requests which involve an easier processing load are given a higher priority value. For example, the request to add the account owner's salary may not involve much client interaction, such as a PIN (personal identification number) checking routine to authenticate the client, since this is a request to add money to an account, not a request to withdraw money. Thus, this request may involve a lighter processing load and should be given a higher priority during a time when the execution threads are heavily loaded.

Detailed Description Text (7):
Another part of the rule could be that a request from a certain client should be given priority over other clients in the queue. For example, the owner of the bank account that is waiting at the ATM machine can be given priority over the direct debit and direct deposit requests. Alternatively, a direct deposit request can be given priority over any debit request so as to increase the chances that there will be enough funds in the account to cover the debits.

Detailed Description Text (10):
Request re-ordering unit 35 then examines the priority values assigned to each of the queued requests and re-orders the queued requests according to their priority values so that the highest priority valued request is at the top of the queue to be next dispatched to an execution thread, and the other requests are placed in descending order according to descending priority values.

Detailed Description Text (11):
It should be noted that the order of the queued requests can be dynamically changed, that is, the order can be changed even after the request re-ordering unit

35 has re-ordered the requests, if the state of the system has changed. For example, if thread 33b suddenly becomes free after the request re-ordering unit 35 has re-ordered the queued requests, priority determining unit 34 now follows a part of the programmed rule that states that if thread 33b becomes free then a computation-intensive request (e.g., the request of the account owner to withdraw funds from the ATM, which involves PIN checking and other client interaction) should be given a high priority value. This may be, for example, that thread 33b is particularly well adapted for handling heavy processing loads, so if it becomes free, an appropriate request should be scheduled as soon as possible for execution on thread 33b in order to provide as efficient a workload balancing amongst threads as possible. In this regard, the frequency with which the priority determining unit applies the rule to its inputs can also be set by the programmer. One choice might be each time a new request is received in the queue. Another may be each time a request is dispatched from the queue. A third choice may be after a predetermined time period (e.g., 5 seconds) has elapsed.

Detailed Description Text (12):
Again, the rule followed by priority determining unit 34 can be programmed to suit the exact concerns of the programmer. For example, the exact levels of priority can be set to give higher priority to a heavy processing request when thread 33b becomes free as compared to an account balance inquiry request, if the programmer decides that it is more important to efficiently balance the workload as compared to giving a most recent account balance to a client.

CLAIMS:

1. An apparatus for scheduling and dispatching client requests for execution by a server object in a heterogeneous object-oriented client/server computing environment, the apparatus comprising: a request-holding buffer having an input connected to a communications channel which channels the client requests to the apparatus, and an output; a plurality of parallel execution threads connected to the output of the buffer, upon each of the plurality of parallel execution threads the server object runs an instance of itself thereby allowing the server object to process a plurality of requests at the same time; and a scheduling means for distributing client requests stored in the buffer to the plurality of execution threads; wherein the scheduling means places client requests held in the buffer in priority order-based on a priority determining rule which takes into account a current processing load of the plurality of execution threads and the nature of each of the held requests.

5. A method of scheduling and dispatching client requests for execution by a server object in a heterogeneous object-oriented client/server computing environment, comprising the steps of: determining information about each of a plurality of queued incoming client requests; determining a current processing load of each of a plurality of parallel execution threads of the server object, wherein upon each of the plurality of parallel execution threads the server object the server object runs an instance of itself thereby allowing the server object to process a plurality of requests at the same time; applying a priority determining rule to the information obtained in said determining steps; and scheduling the order of dispatch from the queue of the plurality of queued requests based on the results of said applying step.

6. The method of claim 5 wherein said queued client requests are stored within an object adapter.

12. The computer program product of claim 10 said queued client requests are stored within an object adapter.

L26: Entry 2 of 6 File: USPT Jan 7, 2003

DOCUMENT-IDENTIFIER: US 6505250 B2
TITLE: Apparatus and method for scheduling and dispatching queued client requests within a server in a client/server computer system

Abstract Text (1):
An apparatus for scheduling and dispatching client requests for execution by a server object in a heterogeneous object-oriented client/server computing environment, the apparatus comprising: a request-holding buffer having an input connected to a communications channel which channels the client requests to the apparatus, and an output; a plurality of parallel execution threads connected to the output of the buffer; and a scheduling means for distributing client requests stored in the buffer to the plurality of execution threads, characterized in that: the scheduling means places client requests held in the buffer in priority order based on a priority determining rule which takes into account the state of the plurality of execution threads and the nature of each of the held requests.

Brief Summary Text (2):
The invention relates to the field of client/server (also known as "distributed") computing, where one computing device ("the client") requests another computing device ("the server") to perform part of the client's work.

Brief Summary Text (5):
The benefits of client/server computing have been even further enhanced by the use of a well-known computer programming technology called object-oriented programming (OOP), which allows the client and server to be located on different (heterogeneous) "platforms". A platform is a combination of the specific hardware/software/operating system/communication protocol which a machine uses to do its work. OOP allows the client application program and server application program to operate on their own platforms without worrying how the client application's work requests will be communicated and accepted by the server application. Likewise, the server application does not have to worry about how the OOP system will receive, translate and send the server application's processing results back to the requesting client application.

Brief Summary Text (8):
When the client computer 10 wishes to make a request for the server computer 20's services, the first application program 40 informs the first logic means 50 of the service required. It may for example do this by sending the first logic means the name of a remote procedure along with a list of input and output parameters. The first logic means 50 then handles the task of establishing the necessary communications with the second computer 20 with reference to definitions of the available communications services stored in the storage device 60. All the possible services are defined as a cohesive framework of object classes 70, these classes being derived from a single object class. Defining the services in this way gives rise to a great number of advantages in terms of performance and reusability.

Brief Summary Text (13):
FIG. 2 shows a conventional architecture for such a system. Once client requests find their way through the ORB 21 and into the server, the ORB finds a particular server object capable of executing the request and sends the request to that server

object's object adapter 22 (also defined by OMG standard) where it is stored in the object adapter's buffer to await processing by the server object. The buffer is a First-In-First-Out queue, meaning that the first request received in the buffer at one end thereof is the first to leave out the other end. The server object has a plurality of parallel execution threads (23a, 23b, 23c) upon any of which it can run an instance of itself. In this way, the server object is able to process plural requests at the same time. The object adapter 22 looks to see which of the parallel execution threads is ready to process another request and assigns the request located at the end of the buffer to the next available execution thread. This is explained in the above-mentioned U.S. Patent as a "dispatching" mechanism whereby the server dispatches queued requests to execution threads.

Brief Summary Text (14):
One major problem with this prior architecture is that it is not possible to obtain a predictable response time for the execution of a client request. That is, a particular client request could be sitting in a server object's object adapter queue 22 behind a large number of other requests, or, at another time, the particular client request could be the only request in the queue. The client that is waiting for an answer cannot predict when a response will be received from the server object. Another problem is that a very important client request may have to wait behind many not so important requests in the object adapter queue.

Brief Summary Text (17):
According to one aspect, the present invention provides an apparatus for scheduling and dispatching client requests for execution by a server object in a heterogeneous object-oriented client/server computing environment, the apparatus comprising: a request-holding buffer having an input connected to a communications channel which channels the client requests to the apparatus, and an output; a plurality of parallel execution threads connected to the output of the buffer; and a scheduling means for distributing client requests stored in the buffer to the plurality of execution threads, characterized in that: the scheduling means places client requests held in the buffer in priority order based on a priority determining rule which takes into account the state of the plurality of execution threads and the nature of each of the held requests.

Brief Summary Text (20):
According to a second aspect, the present invention provides a method of scheduling and dispatching client requests for execution by a server object in a heterogeneous object-oriented client/server computing environment, comprising the steps of: determining information about each of a plurality of queued incoming client requests; determining information about each of a plurality of parallel execution threads of the server object; applying a priority determining rule to the information obtained in said determining steps; and scheduling the order of dispatch from the queue of the plurality of queued requests based on the results of said applying step.

Brief Summary Text (22):
Thus, with the present invention, queued client requests can be processed in a much more efficient and controllable manner, greatly enhancing the predictability of processing result which is returned to the client. High priority client requests can be processed before lower priority requests and workload management amongst the execution threads can be effected, to provide highly efficient and predictable processing of the queued requests.

Detailed Description Text (2):
In the preferred embodiment of FIG. 3, requests received at the server process from client processes are first received by the server's ORB 31. ORB 31 then passes on requests destined to a particular server object to that server object's object adapter 32. This server object has a number of parallel execution threads 33a, 33b and 33c where different instances of the server object can be running in parallel,

in order to execute a large number of client requests. This is all analogous to the prior art of FIG. 2 that was described above.

Detailed Description Text (4):
In the example that will be described hereinbelow to illustrate the operation of this preferred embodiment, the server object will represent a bank account. Thus, the various requests that are queued in object adapter 32 are requests to access a particular bank account. One queued request is from a client ATM (automated teller machine) to withdraw funds from this account. This request is from the person owning the account who wishes to withdraw some funds. A second queued request is from a direct deposit salary payer client. This request is from the account owner's employer and the employer is adding the employer's monthly salary into the account owner's bank account. A third queued request is from another client ATM to check the balance of the account. This request is from the account owner's wife, who is on the other side of town from the owner at another client ATM machine. A fourth queued request is a direct debit request from the electricity company that supplies electricity to the account owner's household. The request is a debit of the account for the amount of the monthly electricity bill.

Detailed Description Text (5):
The priority determining unit 34 operates according to a programmed rule in order to determine the priority of the queued requests in object adapter 32 that are awaiting execution by the server object. For example, one part of the rule is that requests to check the balance of the account should be given a low priority value, since the reply to this request will be more informative to the client if other pending requests are executed first. That is, if a large amount of money is going to be debited from the account by a direct debit, it is better that the person requesting the balance of the account be given the balance after the direct debit rather than before the direct debit. This gives a more current version of the balance to the person requesting the balance.

Detailed Description Text (6):
Another part of the rule is that if threads 33a, 33b and 33c are heavily loaded (are performing a high level of processing as compared to normal) requests which involve an easier processing load are given a higher priority value. For example, the request to add the account owner's salary may not involve much client interaction, such as a PIN (personal identification number) checking routine to authenticate the client, since this is a request to add money to an account, not a request to withdraw money. Thus, this request may involve a lighter processing load and should be given a higher priority during a time when the execution threads are heavily loaded.

Detailed Description Text (7):
Another part of the rule could be that a request from a certain client should be given priority over other clients in the queue. For example, the owner of the bank account that is waiting at the ATM machine can be given priority over the direct debit and direct deposit requests. Alternatively, a direct deposit request can be given priority over any debit request so as to increase the chances that there will be enough funds in the account to cover the debits.

Detailed Description Text (10):
Request re-ordering unit 35 then examines the priority values assigned to each of the queued requests and re-orders the queued requests according to their priority values so that the highest priority valued request is at the top of the queue to be next dispatched to an execution thread, and the other requests are placed in descending order according to descending priority values.

Detailed Description Text (11):
It should be noted that the order of the queued requests can be dynamically changed, that is, the order can be changed even after the request re-ordering unit

35 has re-ordered the requests, if the state of the system has changed. For example, if thread 33b suddenly becomes free after the request re-ordering unit 35 has re-ordered the queued requests, priority determining unit 34 now follows a part of the programmed rule that states that if thread 33b becomes free then a computation-intensive request (e.g., the request of the account owner to withdraw funds from the ATM, which involves PIN checking and other client interaction) should be given a high priority value. This may be, for example, that thread 33b is particularly well adapted for handling heavy processing loads, so if it becomes free, an appropriate request should be scheduled as soon as possible for execution on thread 33b in order to provide as efficient a workload balancing amongst threads as possible. In this regard, the frequency with which the priority determining unit applies the rule to its inputs can also be set by the programmer. One choice might be each time a new request is received in the queue. Another may be each time a request is dispatched from the queue. A third choice may be after a predetermined time period (e.g., 5 seconds) has elapsed.

Detailed Description Text (12):
Again, the rule followed by priority determining unit 34 can be programmed to suit the exact concerns of the programmer. For example, the exact levels of priority can be set to give higher priority to a heavy processing request when thread 33b becomes free as compared to an account balance inquiry request, if the programmer decides that it is more important to efficiently balance the workload as compared to giving a most recent account balance to a client.

CLAIMS:

1. An apparatus for scheduling and dispatching client requests for execution by a server object in a heterogeneous object-oriented client/server computing environment, the apparatus comprising: a request-holding buffer having an input connected to a communications channel which channels the client requests to the apparatus, and an output; a plurality of parallel execution threads connected to the output of the buffer, upon each of the plurality of parallel execution threads the server object runs an instance of itself thereby allowing the server object to process a plurality of requests at the same time; and a scheduling means for distributing client requests stored in the buffer to the plurality of execution threads; wherein the scheduling means places client requests held in the buffer in priority order-based on a priority determining rule which takes into account a current processing load of the plurality of execution threads and the nature of each of the held requests.

5. A method of scheduling and dispatching client requests for execution by a server object in a heterogeneous object-oriented client/server computing environment, comprising the steps of: determining information about each of a plurality of queued incoming client requests; determining a current processing load of each of a plurality of parallel execution threads of the server object, wherein upon each of the plurality of parallel execution threads the server object the server object runs an instance of itself thereby allowing the server object to process a plurality of requests at the same time; applying a priority determining rule to the information obtained in said determining steps; and scheduling the order of dispatch from the queue of the plurality of queued requests based on the results of said applying step.

6. The method of claim 5 wherein said queued client requests are stored within an object adapter.

12. The computer program product of claim 10 said queued client requests are stored within an object adapter.

L31: Entry 2 of 3 File: USPT Apr 9, 1996

DOCUMENT-IDENTIFIER: US 5506969 A
TITLE: Method and apparatus for bus bandwidth management

Abstract Text (1):
A computer system includes bus bandwidth management for operation of a high-speed bus. The high-speed bus is coupled to a plurality of modules. A plurality of client applications operate on the computer system, and request services from the high-speed bus to transfer data from a source module to at least one destination module. The bus bandwidth management system contains a bus manager, a dispatcher, a global controller, and a local controller contained on each module. Transfer requests for data transfer on the high-speed bus are made from the client applications to the bus manager. The bus manager takes the requested information and, based on a bus management policy management in effect, schedules a transfer order for the transfer requests. The bus manager then transfers the ordered transfer requests to the dispatcher. The dispatcher decomposes the ordered transfer requests into individual bus transfer operations. For each individual bus transfer operation, the dispatcher loads a command packet into the global controller, the source module, and the destination module(s). After the dispatcher dispatches all individual bus transfer operations, the dispatcher returns to the bus manager to receive the next transfer request. The global controller executes the individual bus transfer operations in the transfer order.

Brief Summary Text (7):
In recognition that not all clients should receive equal allocation of bus resources, some bus arbitration systems employ static priority policies. In a static priority bus allocation policy, each client application competing for use of the bus has a priority attribute associated with its request for bus services. Typically, the priority attribute consists of a number representing the priority of the request. With use of the static priority policy, when a number of requests are made for bus resources, the bus arbitration system selects the client with the highest priority. Although the static priority provides improved performance over the fairness-based policies for some applications, the static priority policy also fails to account for timeliness characteristics associated with the client requests.

Brief Summary Text (10):
A computer system includes bus bandwidth management for the operation of a high-speed bus. The high-speed bus is coupled to a plurality of modules. A plurality of client applications operate on the computer system, and request services from the high-speed bus to transfer data from a source module to at least one destination module. The bus bandwidth management system contains a bus manager, a dispatcher, a global controller, and a local controller contained on each module. The high-speed bus comprises a high-speed control bus (HSCB), and a high-speed data bus (HSDB). For any particular transfer request by a client application, a source module and at least one destination module are identified. The bus manager is configured to receive transfer requests from the client applications. The bus manager is coupled to a dispatcher, and in turn, the dispatcher is coupled to the global controller. The dispatcher and the global controller operate in conjunction to perform bus arbitration and the dispatching of individual transfer operations. The global controller is coupled to the HSCB for setting up a plurality of bus transfer

operations.

Brief Summary Text (12):
Upon determining the transfer order for the transfer requests, the bus manager transfers the ordered set of transfer requests to the dispatcher. The dispatcher decomposes the ordered transfer requests into individual bus transfer operations. Each individual bus transfer operation contains a command for the global controller, source and destination modules. For each individual bus transfer operation, the dispatcher loads the global controller command into the global controller, the source module command packet into the source module, and the destination module command packet into the destination module(s). After the dispatcher dispatches all individual bus transfer operations for a given transfer request, the dispatcher returns to the bus manager to receive the next transfer request. The global controller executes the commands in its command queue, in the transfer order, to effectuate all individual bus transfer operations.

Detailed Description Text (12):
The bus scheduler 160 schedules transfer requests from the client applications for use of the high-speed bus 100. In a preferred embodiment of the present invention, the bus scheduler is performed in software and is described more fully below. In general, the bus scheduler 160 orders the transfer requests based on a bus management policy using the importance and urgency information provided from each requesting client application. Upon ordering of the transfer requests, the bus scheduler 160 transfers the ordered set of requests to a bus dispatcher 170. The bus dispatcher 170 is coupled to the high-speed bus 100. The bus dispatcher 170 provides a lower level function such that each individual transfer request is dispatched on the high-speed bus 100 in the order scheduled by the bus scheduler 160. In a preferred embodiment of the present invention, the bus dispatcher 170 is performed in both hardware and software.

Detailed Description Text (13):
Referring to FIG. 2, a high-level block diagram of a high-speed bus configured in accordance with the present invention is illustrated. The high-speed bus 100 comprises a high-speed control bus (HSCB) 200 and a high-speed data bus (HSDB) 210. In general, the high-speed bus 100 couples a plurality of modules. For any particular transfer request, a source module and at least one destination module is identified. For the example shown in FIG. 2, source module 215 transfers data over the HSDB 210 to a destination module 220. As will be appreciated by one skilled in the art, any number of modules could be connected to the high-speed bus 100. For the example illustrated in FIG. 2, other modules 225 represents additional modules not involved in the particular data transfer. In a preferred embodiment, the bus management scheduling is performed by a bus manager 230. The bus manager 230 is configured to receive transfer requests from the client applications. The bus manager 230 is coupled to a dispatcher 235, and in turn, the dispatcher 235 is coupled to the global controller 240. The dispatcher 235 and the global controller 240 operate in conjunction to perform bus arbitration dispatching. The global controller 240 is coupled to the HSCB 200 for setting up a plurality of bus transfer operations.

Detailed Description Text (15):
Upon determining the transfer order for the transfer requests, the bus manager transfers the ordered transfer requests to the dispatcher as shown in block 320. The dispatcher retrieves the next logical transfer request, and decomposes the ordered transfer requests into individual bus transfer operations as shown in blocks 325 and 327, respectively. Each individual bus transfer operation contains command packets for the global controller, source and destination modules. A command packet for the source module contains a starting memory address, a count of the number of data words for transfer, and an indication that the transfer operation is a read operation. The destination module command packet contains a starting memory location, a count of the number of words that the destination

module receives, and an indication that the transfer operation is a write operation. Also, the global controller command packet contains the source and destination module identifiers.

Detailed Description Text (16):
In order to generate the command packets, the dispatcher utilizes the source and destination rectangular descriptions provided by the bus manger. For each individual bus transfer operation, the dispatcher loads the global controller command packet into the global controller, the source module command packet into the source module, and the destination module command packet into the destination module(s) as shown in block 330. As illustrated in block 340, if all the individual bus transfer operations are not dispatched, then the dispatcher loads the command queue of the global, source and destination modules for next bus transfer operation. Alternatively, if all the individual bus transfer operations are dispatched, then the dispatcher determines whether all logical transfer requests are serviced as shown in block 350. If all logical transfer requests are not serviced, then the dispatcher retrieves the next logical transfer request. As shown in block 360, the global controller executes its command queue, in the transfer order, to effectuate all individual bus transfer operations as will be described more fully below.

Detailed Description Text (18):
The time constraint information of the bus transfer request message includes urgency information for the transfer request specified relative to a real-time clock of the computer system. In general, the urgency information specifies a deadline time for completion of the bus operation. For time critical client application programs, the deadline or urgency information contains the time limitations associated with the time critical application. In addition to the urgency information, the time constraint information of the bus transfer request message includes importance information. The importance information provides a user-defined priority for the transfer request specified as a global priority.

Detailed Description Text (20):
In a preferred embodiment of the present invention, the bus scheduler utilizes a time driven resource management (TDRM) policy to allocate bus resources. Referring to FIG. 5, a flow diagram for a TDRM bus resource management policy is illustrated. In order to implement the TDRM policy, the bus scheduler maintains an importance list and an urgency order list for incoming transfer requests. Upon receipt of each transfer request, the bus scheduler inserts the transfer request on the importance order list, based on the global priority of the transfer request, and inserts the transfer request on the urgency order list based on the deadline information received as shown in blocks 510, 520 and 530, respectively. In accordance with the TDRM policy, the bus scheduler determines if the bus bandwidth is adequate to service all pending transfer requests within the specified deadline as shown in block 540. To accomplish this task, the bus scheduler constructs a trial comprising deadline-ordered schedule of transfer requests. For the trial, the bus scheduler assumes no additional transfer requests are generated by a client application program, and times calculated for all pending transfer requests are determined based on the size of the transfer request.

Detailed Description Text (21):
If the bus bandwidth is adequate, then the bus scheduler utilizes the urgency order list as the order for the logical transfer requests as shown in block 550. Subsequently, the bus scheduler issues the ordered logical transfer requests to the dispatcher. Alternatively, if the bus bandwidth is not adequate to service all pending transfer requests within the corresponding deadlines, then the bus scheduler re-orders the urgency order list by removing the least important transfer request as shown in block 560. Specifically, the least important transfer request is placed at the bottom of the urgency order list. If the bus scheduler determines that the bus bandwidth is sufficient to service the remaining transfer requests on

the re-ordered urgency list, then the bus scheduler dispatches the logical transfer requests to the dispatcher based on the newly reordered urgency list. Otherwise, the process, illustrated by blocks 540, 550 and 560, is executed until a logical transfer request list is generated that permits servicing of the largest number of the most important transfer requests. After the bus scheduler issues the logical transfer requests to the dispatcher, the bus scheduler waits for a response from the dispatcher that indicates completion or failure of the bus operations. The bus scheduler then sends a reply message to the specified client application program, removes the corresponding transfer request from the urgency and importance lists, and instructs the dispatcher to execute the next transfer.

Detailed Description Text (22):
Referring to FIG. 6, a block diagram of a global bus controller and local controllers for source and destination modules configured in accordance with the present invention is illustrated. For purposes of explanation, a local controller 605 for a single source/destination module is illustrated in FIG. 6. However, each module coupled to the high-speed bus 100 comprises a local controller such as local controller 605. The local controller 605 in each source/destination module contains a command queue 610, an address and count registers 628 and 630, a memory 626, and local control logic 632. The command queue 610 is arranged as a first-in-first-out (FIFO) device. The command queue 610 stores the source/destination command packets transferred from the dispatcher 235. Because the command queue 610 is arranged as a FIFO device, the source and destination command packets are stored in the transfer order.

Detailed Description Text (23):
Also shown in FIG. 6 is a global controller 600. The global controller 600 contains a command queue 610, also coupled to the dispatcher 235 through the HSCB 200. The command queue 615 is also a FIFO device such that global controller command packets are stored in the order dispatched (i.e. the transfer order). The global controller 600 also contains source and destination registers 619 and 617, respectively, and global control logic 621. The command queue is coupled to the source and destination registers 619 and 617 to store source and destination identifiers for each bus transfer operation. The global control logic 621 is coupled to the HSCB 200, and generates signals to effectuate bus transfer operation. The global control logic 621 comprises a high-speed bus clock (HSBCLK) for providing timing for the HSDB 210. In a preferred embodiment, the clock comprises a 20 megahertz (MHz) frequency. The operation of the global control logic 621 is described more fully below.

Detailed Description Text (24):
Through the command queues on the local and global controller modules, the present invention provides for set-up of N bus transfer operations, wherein N is greater than one. As discussed above, the command packets are dispatched on HSCB 200. In this way, the present invention overlaps the set-up of bus operations over the HSCB 200 with the transfer of data. The ability to set up N bus transfer operations is quantitatively better than merely setting up one transfer. The setting up of N bus transfer operations permits the high level intelligent bus bandwidth management of the present invention. Typically, a bus exhibits up to 50% reduction in bus bandwidth resulting from time required for set-up which translates into idle time on the data transfer bus. In a preferred embodiment of the present invention, the command queues of the local and global controller modules support up to 64 bus transfer operations. In addition, mechanisms exist to indicate the degree of fullness of the queues so that the software controls over-running and under-running into the queues.

Detailed Description Text (30):
In the computer system 900, the high-speed bus 911 couples a plurality of I/O devices 925, 930, and 935 contained in an I/O subsystem 920. The I/O devices may perform a variety of functions requiring high speed data transfer. For example, the

high-speed bus of the present invention has application for high speed data transfer to accommodate time-critical media applications. Each I/O device comprises a local controller. In addition, I/O device 935 comprises a global controller. As described above, commands are dispatched on the HSCB 913 to set-up a plurality of bus transfer operations. The bus transfer operations are executed, in the transfer order, by the global controller contained on the I/O device 935 such that data is transferred on the HSDB 912.

CLAIMS:

1. A method for bus bandwidth management comprising the steps of:

issuing transfer requests from a plurality of client applications operating on a computer system to effectuate data transfer on a bus, each of said transfer requests specifying urgency and importance information, wherein said urgency information specifies a time deadline for execution of said transfer request and said importance information specifies a priority for said transfer request;

scheduling said transfer requests by ordering said transfer requests, based on a bus management policy that utilizes said urgency and importance information, to generate a transfer order;

dispatching said transfer requests in said transfer order to a command queue by generating individual bus operations; and

transferring data via said bus for each individual bus operation in said transfer order.

3. The method as claimed in claim 1, wherein scheduling said transfer requests for use of said bus based on a bus management policy comprises the steps of:

generating an importance list and an urgency list for each transfer request based on said urgency and importance information;

determining whether said bus comprises enough bus bandwidth to service said transfer requests on said urgency list within said deadline specified for each transfer request;

generating said transfer order corresponding to said urgency list when said bus comprises enough bus bandwidth to service said transfer requests; and

re-ordering said urgency list, when said bus does not comprises enough bus bandwidth, to service said transfer requests by removing transfer requests comprising low priorities until said bus comprises enough bus bandwidth to service said remaining transfer requests.

4. The method as claimed in claim 1 wherein the step of dispatching said transfer requests comprises the steps of:

providing a control bus; and

dispatching said transfer requests on said control bus to said command queue.

7. The method as claimed in claim 6 wherein the step of dispatching said individual bus operations in said transfer order comprises the steps of:

decomposing said transfer requests into individual bus operations, each individual bus request comprising source, destination and global command packets, said source and destination command packets comprising a starting memory location, word count, read/write indication, and said global command packet comprising source and

destination identifiers; and

transferring, for each individual bus operation, said source command packet to a queue in said local controller on said source module, said destination command packet to a queue in said local controller on said destination module, and said global command packet to a queue in said global controller.

8. In a computer system comprising a high-speed data bus coupled to a plurality of modules, An apparatus for bus bandwidth management comprising:

a plurality of client applications operating on a computer system for issuing transfer requests to a bus, each of said transfer requests specifying urgency and importance information, wherein said urgency information specifies a time deadline for execution of said transfer request and said importance information specifies a priority for said transfer request;

a bus manager coupled to said client applications for scheduling said transfer requests for dispatch on said bus, said bus manager for receiving said transfer requests and for ordering said transfer requests, based on a bus management policy that utilizes said urgency and importance information, to generate a transfer order;

a command queue;

a bus dispatcher coupled to said bus manager for dispatching said transfer requests in said transfer order to said command queue, and for generating individual bus operations; and

a global controller coupled to said command queue for transferring data in said transfer order via said for each individual bus operation.

10. The apparatus as claimed in claim 8, wherein said bus manager comprises a Time Driven Resource Management (TDRM) policy manager comprising:

a list generator coupled to said client applications for generating an importance list and an urgency list for each transfer request based on said urgency and importance information;

a bus bandwidth analyzer for determining whether said bus comprises enough bus bandwidth to service said transfer requests on said urgency list within said deadline specified for each transfer request; and

transfer request ordering coupled to said bus bandwidth analyzer and said list generator for generating said transfer order corresponding to said urgency list when said bus comprises enough bus bandwidth to service said transfer requests, said transfer request ordering for re-ordering said urgency list, when said high-speed bus does not comprises enough bus bandwidth, and for servicing said transfer requests by removing transfer requests comprising low priorities until said high-speed bus comprises enough bus bandwidth to service said remaining transfer requests.

11. The apparatus as claimed in claim 8, wherein said bus dispatcher comprises a control bus for dispatching said transfer requests on to said command queue.

14. The apparatus as claimed in claim 13, wherein said bus dispatcher further comprises:

a command decomposer for decomposing said transfer requests into individual bus operations, each individual bus request comprising source, destination and global command packets, said source and destination command packets comprising a starting

memory location, word count, read/write indication, and said global command packet comprising source and destination identifiers; and

a queue coupled to said command decomposer for transferring, for each individual bus operation, said source command packet to a queue in said local controller on said source module, said destination command packet to a queue in said local controller on said destination module, and said global command packet to a queue in said global controller.

15. A computer system comprising:

at least one central processing unit (CPU);

a plurality of modules;

a bus coupling said at least one CPU to said modules;

a plurality of client applications operating on said computer system that issue transfer requests to said bus, each of said transfer requests identifying a source module and a destination module including specifying urgency and importance information for each transfer request, wherein said urgency information specifies a time deadline for execution of said transfer request and said importance information specifies a priority for said transfer request;

a bus manager coupled to said client applications for scheduling said transfer requests for dispatch on said bus, said bus manager for receiving said transfer requests and for ordering said transfer requests, based on a bus management policy that utilizes said urgency and importance information, to generate a transfer order;

a command queue;

a bus dispatcher coupled to said bus manager for dispatching said transfer requests in said transfer order to said command queue, said bus dispatcher for receiving said transfer requests in said transfer order and for generating individual bus operations; and

a global controller coupled to said command queue for transferring data from said source module to at least one destination module in said transfer order via said for each individual bus operation.

17. The computer system as set forth in claim 15, wherein said bus manager comprises Time Driven Resource Management (TDRM) policy manager comprising:

a list generator coupled to said client applications for generating an importance list and an urgency list for each transfer request based on said urgency and importance information;

a bus bandwidth analyzer for determining whether said bus comprises enough bus bandwidth to service said transfer requests on said urgency list within said deadline specified for each transfer request; and

transfer request ordering coupled to said bus bandwidth analyzer and said list generator for generating said transfer order corresponding to said urgency list when said bus comprises enough bus bandwidth to service said transfer requests, said transfer request ordering for re-ordering said urgency list, when said bus does not comprises enough bus bandwidth, to service said transfer requests by removing transfer requests comprising low priorities until said bus comprises enough bus bandwidth to service said remaining transfer requests.

18. The computer system as set forth in claim 15, wherein said bus dispatcher comprises a control bus dispatching said transfer requests on to said command queue.

21. The computer system as set forth in claim 20 wherein said bus dispatcher further comprises:

a command decomposer for decomposing said transfer requests into individual bus operations, each individual bus request comprising source, destination and global command packets, said source and destination command packets comprising a starting memory location, word count, read/write indication, and said global command packet comprising source and destination identifiers; and

a queue coupled to said command decomposer for transferring, for each individual bus operation, said source command packet to a queue in said local controller on said source module, said destination command packet to a queue in said local controller on said destination module, and said global command packet to a queue in said global controller.

L39: Entry 15 of 27 File: USPT Aug 31, 1999

DOCUMENT-IDENTIFIER: US 5946498 A
TITLE: Delivery of client remote procedure calls to a server via a request queue utilizing priority and time-out

Abstract Text (1):
A computer system includes at least a server procedure and a client processor. The client processor includes plural applications which issue requests for execution of server procedures. A queue is present in the client processor and lists the requests. The client processor also includes plural input/output procedures which enable dispatch of requests to the server procedure. The client processor operates in conjunction with the I/O procedures, removes a request from the queue, and dispatches the request to the server procedure. The processor responds to occurrence of a time-out, with no response being received from the server procedure, to place the current request at the beginning of the queue so as to enable the I/O procedure to service a further request from the queue. The I/O procedure of the client processor handles requests from the queue, based upon assigned priority levels, and removes from the queue a highest assigned priority request that is closest to the end of the queue, before removing a lower priority request.

L39: Entry 11 of 27 File: USPT Dec 5, 2000

DOCUMENT-IDENTIFIER: US 6157963 A
TITLE: System controller with plurality of memory queues for prioritized scheduling of I/O requests from priority assigned clients

Abstract Text (1):
A system for globally prioritizing and scheduling I/O requests from a plurality of storage users or clients to one or more storage objects. The system comprises a storage controller configured to receive I/O requests from the client workstations and prioritize and schedule those I/O requests in accordance with a scheduling algorithm. Specifically, the storage controller receives I/O requests from the storage users and places the I/O requests in memory queues associated with the particular storage users. The storage controller then selects the I/O requests from the various memory queues based on the scheduling algorithm.

L39: Entry 11 of 27 File: USPT Dec 5, 2000

DOCUMENT-IDENTIFIER: US 6157963 A
TITLE: System controller with plurality of memory queues for prioritized scheduling of I/O requests from priority assigned clients

Abstract Text (1):
A system for globally prioritizing and scheduling I/O requests from a plurality of storage users or clients to one or more storage objects. The system comprises a storage controller configured to receive I/O requests from the client workstations and prioritize and schedule those I/O requests in accordance with a scheduling algorithm. Specifically, the storage controller receives I/O requests from the storage users and places the I/O requests in memory queues associated with the particular storage users. The storage controller then selects the I/O requests from the various memory queues based on the scheduling algorithm.

L39: Entry 6 of 27 File: USPT Aug 27, 2002

DOCUMENT-IDENTIFIER: US 6442550 B1
TITLE: System and method in a collaborative data processing environment for customizing the quality of service on a per-client basis

Abstract Text (1):
A server within a collaborative data processing environment transmits a priority value to a first client in response to a first request from the first client. A second request from the first client, which includes that priority value, and a third request from a second client are then received by the server. Thereafter, the server provides the second request with preferential treatment, relative to the third request, based on the priority value of the second request. In an illustrative embodiment, the priority value is transmitted to the first client within a Hypertext Transfer Protocol (HTTP) cookie.

L39: Entry 3 of 27 File: USPT May 13, 2003

DOCUMENT-IDENTIFIER: US 6563506 B1
TITLE: Method and apparatus for memory bandwith allocation and control in a video graphics system

Abstract Text (1):
A method and apparatus for allocation and control of memory bandwidth within a video graphics system is accomplished by first determining the memory bandwidth needs of each of the plurality of memory clients in the video graphics system. Based on this determination, a plurality of timers are configured, wherein each of the timers corresponds to one of the plurality of memory clients. The timers associated with the memory clients store two values. One value indicates the memory access interval for the corresponding client, which determines the spacing between memory access requests that can be issued by that particular client. The other value stored in the time is a memory access limit value, which determines the maximum length of a protected access to the memory by that particular client. A memory controller in the system receives requests from the plurality of clients and determines the priority of the different requests. The memory controller grants the requests with the highest priority, which may result in the termination of a current memory access. However, if the current memory access has equal or greater priority than a received access, the current memory access will not be terminated until the time associated with its memory access limit has expired.